SYSTEM AND METHOD FOR USING A SPLIT-DIRECTORY STRUCTURE FOR SOFTWARE DEVELOPMENT

Inventors: Rob Woollen
Mark Griffith

Claim of Priority:

[0001] This application claims priority to U.S. Provisional Patent Application 60/450,781, filed February 28, 2003 entitled "SYSTEM AND METHOD FOR USING A SPLIT-DIRECTORY STRUCTURE FOR SOFTWARE DEVELOPMENT" (Atty. Docket No. BEAS-01433US0); and U.S. Patent Application No. _____, filed February _____, 2004 entitled "SYSTEM AND METHOD FOR USING A SPLIT-DIRECTORY STRUCTURE FOR SOFTWARE DEVELOPMENT" (Atty. Docket No. BEAS-01433US1), both of which are incorporated herein by reference.

Field of the Invention:

[0002] The invention relates generally to software development, and particularly to a system and method for using a split-directory structure for software development.

Background:

[0003] The J2EE specification defines the Enterprise Archive (EAR) file as a standard structure for developing and packaging J2EE applications. EARs are useful for application development, in that the application may, for example, include both a Web application (Webapp) and an Enterprise Java Bean (EJB), which will be packaged into the EAR. However, while this works well for completed applications, it isn't very convenient for application development.

[0004] In a typical application server development environment that uses EARs, a structure similar to that shown in **Figure 1** is often used. As shown in **Figure 1**, the directories for the /myapp application include, for example, a /myejb subdirectory, for storing both ejb source files and XML descriptors. Similarly /myapp may include a /webapp directory, for storing JSPs, html files, and images, etc. All of this code is stored in the source control system for use by the developer in building the application.

[0005] To deploy the application, a number of steps must be performed, namely compiling the Java files, generating any servlets or container classes, and packaging the whole lot in an EAR. The EAR adheres to a format wherein the top level includes an EAR descriptor /META-INF/application.xml, and all of the other components are listed underneath. This approach works well when the application has been fully completed and ready for installation in the production environment. However it's less useful for application development.

[0006] A problem with the traditional approach is that the developer will have both source files and system-generated output in the same directory. This can be confusing. It's also hard to generate a clean build of the application since it's difficult to delete only the system-generated files. A mechanism that allows application development while maintaining some separation between the user-coded and system-generated files would be useful in addressing these problems.

Summary:

[0007] An embodiment of the invention provides a development-oriented directory structure that can be used with an application server and which solves a number of the problems associated with traditional Enterprise Archive (EAR) files. The development-oriented directory structure avoids the copying of files during the build process, and presents a clean separation between human-readable source files stored in a source control system and generated java class files. The two directories (source and output) appear like a single application.

Brief Description of the Figures:

[0008] **Figure 1** shows an illustration of a directory structure used to store files for building an application.

[0009] **Figure 2** shows an illustration of a development split directory structure in accordance with an embodiment of the invention.

Detailed Description:

[0010] An embodiment of the invention provides a development-oriented directory structure that can be used with an application server (for example the WebLogic Server from BEA Systems, Inc.), and which solves a number of the problems associated with traditional Enterprise Archive (EAR) files. The development-oriented directory structure avoids the copying of files during the build process, and presents a clean separation between human-readable source files stored in a source control system and generated java class files. The two directories (source and output) appear like a single application.

[0011] In a traditional software development system, use is made of a separate source directory (/source) for storing source code, typically under management or

control of a source control system, and a build directory (/build) into which Java files, etc, are compiled. Each time a change is made to a small portion of the code, it's necessary to do a full redeployment. The server receiving the build only sees the /build directory, and not the /source directory. Besides being inelegant, this approach prevents the use of Web-editing-in-place of the source code so as to immediately see the effect of changes to the underlying source.

[0012] The solution provided by an embodiment of the invention is to provide a split-directory structure, and modify the server so that it can see both the /source and the /build directories.

[0013] **Figure 2** illustrates an example of a development split-directory structure in accordance with an embodiment of the invention, as it may be interpreted by a server. As shown in **Figure 2**, the /source and /output directories are interpreted as a single directory. This approach requires no copying, in that the server can read source files (e.g. JSP's, XML descriptors, html images, etc.) directly from the split directory structure, without having to copy them to a /build directory. Everything that is generated goes into an /output directory. The benefit to this approach is that, since the server looks at both the directories, the developer can change the source files and the server will see the updates.

Implementation Detail

[0014] The split-directory system provides a recommended directory layout and an accompanying set of ant tasks for building and deploying applications. The split-directory differs from traditional EAR files because it is optimized for iterative development. Instead of a single archived EAR file or an "exploded" EAR directory structure, the split-directory has 2 parallel directories. The source directory contains all of the source files e.g. java files, descriptors, JSP files, HTML etc in a traditional EAR-like directory structure. All generated or compiled classes are stored in a

separate output directory. This arrangement produces several advantages over the traditional exploded EAR file:

1. No need for file copying.
2. Web files can be changed and redeployed in place without rebuilding.
3. Clean separation between source and generated files allows cleaning the build by just removing the output directory.

Server Support for Split-Directory

[0015] The user specifies the output directory when deploying the application to the server. The server automatically recognizes the deployment as a split-directory and uses information in the output directory to locate the source directory. The Server views the application as a union of the source and output directories. When the server searches for a resource or class, it first checks the source directory. If the class or resource is not found, it then checks the output directory.

[0016] When the application is finally deployed, the split-directory appears just as it did using traditional source control. In accordance with one embodiment, the output directory includes a file (build.txt, or in some instances referred to as .beabuild.txt) the content of which indicates that the output directory is in fact an output directory of a corresponding source directory. In this way, other application can be pointed to the appropriate output directory and can still access the source directory when necessary.

[0017] In accordance with one embodiment, the split directory feature can be provided through an abstraction in the server called a virtual JAR file. The virtual JAR file provides an abstraction over the two split directories. When a request is made to the JAR to retrieve e.g. a/b.jar, the virtual JAR checks the /source directory, then checks the /output directory.

Ant Tasks

wlcompile

[0018] wlcompile is the main build task which compiles the application's java files. wlcompile begins by analyzing all of the components and determining their type. It treats components as either EJBs, Web applications, or Java components. Java components are compiled first into the output directory/APP-INF/classes so they will be visible throughout the application. Next, wlcompile builds the EJBs and automatically includes the previously built java components in the compiler's classpath. This allows the EJB components to call the java components without requiring the user to manually edit their classpath. Finally the .java files in the webapp are compiled with the EJB components and java components in the compiler's classpath. This allows the Web Applications to refer to the EJB and application java classes without manually editing the classpath. The following example compiles the application with sources in /acme/myapp to output directory /build/myapp:

```
<wlcompile srcdir="/acme/myapp" destdir="/build/myapp" />
```

[0019] By default wlcompile builds the entire application. It is also possible to instruct wlcompile to build a single component. This is especially useful when developing large applications where the developer is making isolated changes and wants a faster build. When wlcompile builds a single component, it only rebuilds that component. wlcompile does not track dependencies between components so it will not rebuild any other components which depend on the rebuilt component. The following example builds only the AcmeEJB component:

```
<wlcompile srcdir="/acme/myapp" destdir="/build/myapp"
           includes="AcmeEJB" />
```

[0020] As shown in the example above, wlcompile automatically guesses the

type of each component in the application. In general, this guessing works well, but in some cases, especially when components are incomplete, wlcompile cannot accurately determine the type.

[0021] The common case where wlcompile fails is when an EJB is being developed. Users will typically compile their java files first, get this working, and then write their deployment descriptors. wlcompile is unable to determine that this component is an EJB when it does not have descriptors yet. wlcompile allows users to specify a component's type in cases where the guessing is not correct. The following example builds only the AcmeEJB component and hardcodes it to be an EJB:

```
<wlcompile srcdir="/acme/myapp" destdir="/build/myapp">
  <component name="AcmeEJB" type="EJB" />
</wlcompile>
```

Exclude lists may also be specified, so as to build everything but a few portions of the application.

wldeploy

```
<wldeploy user="system" password="gumby1234"
          action="deploy" source="/build/myapp" />
```

wlpackage

The following examples package split-directory myapp as a traditional EAR file:

```
<wlpackage toFile="/acme/App.ear" srcdir="/acme/myapp"
           destdir="/build/myapp" />
```

```
<wlpackage toDir="/acme/explodedEar" srcdir="/acme/myapp"
           destdir="/build/myapp" />
```

Split Directory Recipes

Stand-alone webapp

```
AcmeWeb/WEB-INF/web.xml
AcmeWeb/WEB-INF/weblogic.xml
AcmeWeb/WEB-INF/src/com/acme/web/MyServlet.java
AcmeWeb/login.jsp
AcmeWeb/graphics/logo.jpg
```

[0022] The webapp source is contained within a directory (AcmeWeb). As usual, the descriptors are in the WEB-INF directory. The WEB-INF/src directory is a place for java classes such as servlets which will be compiled to the WEB-INF/classes directory. JSP files, HTML, and other web files are contained under the directory root.

Stand-alone ejbapp

```
AcmeEJB/META-INF/ejb-jar.xml
AcmeEJB/META-INF/weblogic-ejb-jar.xml
AcmeEJB/com/acme/ejb/MyLocal.java
AcmeEJB/com/acme/ejb/MyHome.java
AcmeEJB/com/acme/ejb/MyEJB.java
```

[0023] The ejbapp source is contained within a directory (AcmeEJB). As usual, the descriptors are in the META-INF directory. Any java files under the AcmeEJB root directory are compiled into the output directory.

Application with webapp and ejb

```
InventoryApp/META-INF/application.xml
InventoryApp/META-INF/weblogic-application.xml

InventoryApp/AcmeEJB/META-INF/ejb-jar.xml
InventoryApp/AcmeEJB/META-INF/weblogic-ejb-jar.xml
InventoryApp/AcmeEJB/com/acme/ejb/MyLocal.java
```

```
InventoryApp/AcmeEJB/com/acme/ejb/MyHome.java
InventoryApp/AcmeEJB/com/acme/ejb/MyEJB.java

InventoryApp/AcmeWeb/WEB-INF/web.xml
InventoryApp/AcmeWeb/WEB-INF/weblogic.xml
InventoryApp/AcmeWeb/WEB-INF/src/com/acme/web/MyServlet.java
InventoryApp/AcmeWeb/login.jsp
InventoryApp/AcmeWeb/graphics/logo.jpg
```

[0024] This example shows an Inventory application which is made up of an EJB (AcmeEJB) and a webapp (AcmeWeb). The application's descriptors are contained in the META-INF/ directory as usual. The application components, AcmeEJB and AcmeWeb, are contained in directories directly under the InventoryApp.

Third-Party Jar Files

```
InventoryApp/APP-INF/lib/xmlparser.jar
InventoryApp/APP-INF/lib/customlib.jar
```

[0025] This example demonstrates how the Inventory Application can be extended to use third-party jar files. Third-party jar files are jar files that are packaged with the application, but are usually not the develoer's own code, and are not recompiled. For instance, XML parsers, logging implementations, and other utility jar files are common. These files may be placed in the APP-INF/lib/ subdirectory. They will be visible throughout the application.

Java Utility Classes

```
InventoryApp/AcmeMainframe/com/acme/mainframe/Connect.java
InventoryApp/AcmeMainframe/com/acme/mainframe/Login.java
InventoryApp/AcmeDebug/com/acme/debug/Tracer.java
```

[0026] This example demonstrates how the Inventory Application can be extended to use java utility classes. Java Utility classes differ from third-party jars

because the source files are part of the application and must be compiled. Java Utility classes are typically libraries used by application components such as EJBs or webapps. The build process compiles these files into the APP-INF/classes directory under the output directory. These classes are avaliable throughout the application.

EJB Using EJBGen

```
AcmeEJB/com/acme/ejb/MyEJB.ejb
AcmeEJB/com/acme/ejb/SupportingClass.java
```

[0027] The ejbapp source is contained within a directory (AcmeEJB). The EJBGen source file must have a .ejb extension. The build system runs EJBGen on this file creating the java sources and descriptors in the output directory. Any java files in the source directory will be compiled as usual to the output directory.

[0028] The present invention may be conveniently implemented using a conventional general purpose or a specialized digital computer or microprocessor programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0029] In some embodiments, the present invention includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash

memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0030] The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. Particularly, it will be evident that while the examples described herein illustrate how the split directory feature may be used in a WebLogic environment, other application servers, computing environments, and software development systems may use and benefit from the invention. The code examples given are presented for purposes of illustration. It will be evident that the techniques described herein may be applied using other code languages, and with different code.

[0031] The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.